Aplicação de Redes Neurais no reconhecimento de letras em placas de veículos automotores brasileiros

Herman P. Lima Jr
Aline da Rocha Gesualdi
Marcelo Portes de Albuquerque
Márcio Portes de Albuquerque
Israel Andrade Esquef

Centro Brasileiro de Pesquisas Físicas – CBPF
Rua Dr. Xavier Sigaud, 150
22290-180 – Rio de Janeiro, RJ

RESUMO

Este trabalho tem como objetivo a implementação de uma rede neural artificial sem realimentação (‘feedforward’) que realize o reconhecimento de caracteres alfabéticos obtidos através de fotos digitais de placas de veículos automotores. Será utilizado o método de Retropropagação do erro (‘Backpropagation’) para o treinamento da rede visando a convergência para um erro médio quadrático mínimo. Ao longo do trabalho serão testadas várias configurações de parâmetros de operação da rede tendo como objetivo a melhor eficiência nos resultados.

ÍNDICE

Introdução Teórica

As Redes Neurais ou, mais precisamente Redes Neurais Artificiais, representam uma ferramenta computacional de aplicação nas mais diversas áreas da ciência e da vida humana. Dentre as aplicações mais comuns que encontramos para as redes neurais, podemos destacar: simulação de sistemas não-lineares, controle de plantas industriais e reconhecimento de padrões. Atributos importantes e únicos desta poderosa ferramenta são: aproximação universal (mapeamento entrada-saída não-linear), capacidade de aprender e se adaptar ao ambiente em que está operando e capacidade de generalização.

As redes neurais utilizam um modelo matemático que se baseia originalmente na estrutura neural dos organismos inteligentes. Estas estruturas são formadas por milhões de neurônios interconectados por ligações chamadas sinapses. Estas ligações, além de conduzirem a informação de um neurônio para outro, também possuem pesos que multiplicam a informação aplicada à sua entrada. Com isso, uma informação pode ser atenuada, amplificada ou simplesmente permanecer inalterada quando passada de um neurônio a outro. Estudos científicos já demonstraram que a memória da estrutura neural não está nos neurônios, mas sim nos pesos das sinapses. Estes pesos é que são alterados ou não quando recebem mais informação ou quando estão em processo de aprendizagem, por exemplo.

O neurônio artificial

O neurônio artificial é uma unidade de processamento que, basicamente realiza o somatório de todos os sinais aplicados à sua entrada e aplica este resultado em uma função não-linear, chamada função de ativação. A saída do neurônio é única e é igual ao valor da função de ativação num determinado instante. A estrutura que representa o neurônio artificial é mostrada abaixo.

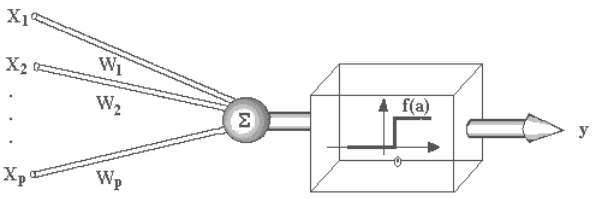


Figura 1: Neurônio Artificial

O vetor de entradas é representado por **X** = [X_1 X_2 ... X_p], as sinapses são representadas pelo vetor de pesos **W** = [W_1 W_2 ... W_p], **f**(a) é a função de ativação e **y** é a saída do neurônio.

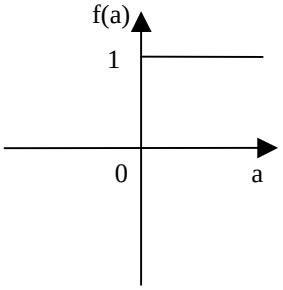
Tipos de função de ativação

A função de ativação define a saída do neurônio em termos do nível do sinal aplicado à sua entrada. Podemos identificar três tipos básicos de funções de ativação:

1. Função 'Threshold' ou lógica

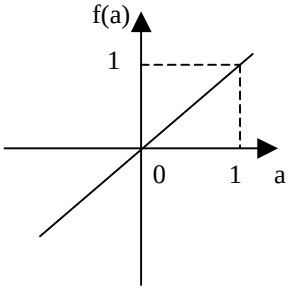
f(a) = 1, se $a \geq 0$ Gráfico:

0, se $a \leq 0$



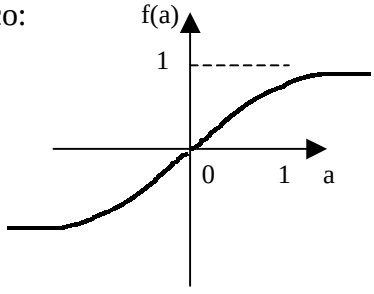
2. Função linear

f(a) = a Gráfico:



3. Função Tangente Hiperbólica

f(a) = tgh(a) Gráfico:



Estrutura da Rede Neural

Uma rede neural é estruturada em camadas de neurônios. Para problemas simples uma camada consegue realizar o mapeamento desejado. Para problemas mais complexos, duas camadas se faz necessário. Em torno de 98% dos problemas práticos, uma rede de duas camadas é suficiente. No caso da rede 'feedforward', ou seja, sem realimentação, a saída de um neurônio pertencente a uma camada é ligada a todos os neurônios da camada posterior, como mostra a figura 2.

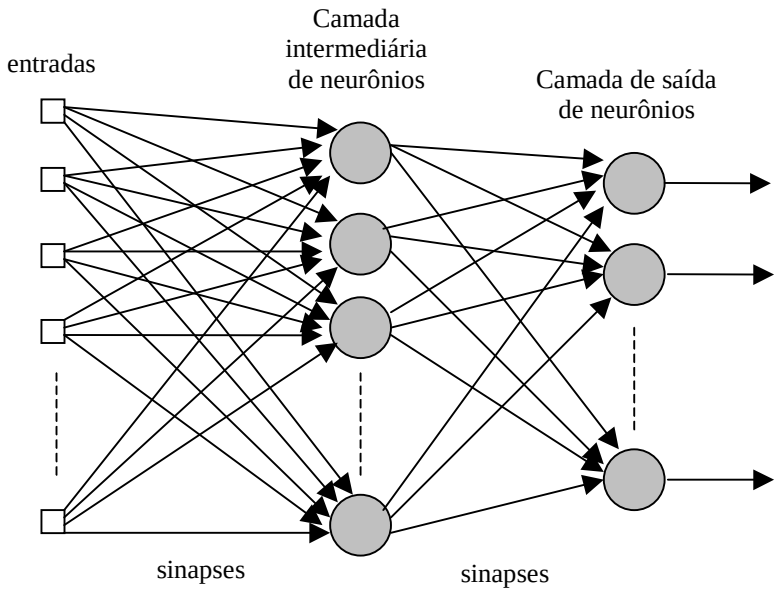


Figura 2: Rede Neural sem realimentação de duas camadas

Algoritmo de 'Backpropagation'

O processo de treinamento de uma rede neural consiste basicamente em apresentar pares entrada-saída à rede, calcular as saídas da rede em função das entradas, calcular o erro entre a saída desejada e a saída calculada e alterar os valores sinápticos por algum tipo de algoritmo. À medida que os pares entrada-saída vão sendo apresentados e os pesos das sinapses atualizados, o erro médio quadrático (MSE) da saída da rede tende a diminuir. Existem alguns tipos mais importantes de algoritmos, dentre os quais o mais utilizado e eficiente é o chamado de 'Backpropagation'.

O processo de treinamento da rede neural pelo método de 'Backpropagation' realiza, como o próprio nome diz, a retropropagação dos erros calculados das saídas em direção às entradas. O erro calculado na saída de um neurônio é multiplicado pela derivada da função de ativação daquele neurônio e propagado para a sua entrada. Este valor então é enviado para todos os neurônios da camada anterior pesado pelas respectivas sinapses.

O algoritmo de 'Backpropagation' utiliza o método do Gradiente Descendente para corrigir os valores sinápticos. Neste método, calculamos o gradiente da função objetivo (MSE) a se minimizar e atualizamos o valor das sinapses somando-se a ele o valor do gradiente com o sinal invertido. O gradiente, neste caso, é a derivada parcial da função erro médio quadrático em relação à sinapse. Este processo faz com que o erro médio quadrático do problema "caminhe" sempre no sentido contrário de seu máximo

crescimento. Resumindo, podemos dizer que a equação básica do Algoritmo de 'Backpropagation' é:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla_k(F_o)$$

onde: $\mathbf{w}_{k+1}$ é o vetor de pesos sinápticos no instante k+1;

$\mathbf{w}_k$ é o vetor de pesos sinápticos no instante k;

α_k é uma constante chamada Passo de treinamento;

$\nabla_k(F_o)$ é vetor gradiente da função objetivo.

O passo de treinamento é uma constante que controla a velocidade de convergência do algoritmo, ou seja, a rapidez com que os valores sinápticos levam o MSE para um mínimo local ou global na superfície de erro do problema. Para pequenos valores de α, a atualização das sinapses é mais lenta, fazendo com que se percorra com maior precisão a superfície de erro do problema. Para valores altos de α, apesar da convergência ocorrer de maneira mais rápida, pode acontecer oscilações no treinamento, podendo ás vezes até impossibilitar a convergência para um mínimo local ou global. Maiores detalhes do Algoritmo de 'Backpropagation' podem ser encontrados na referência [1], já que não faz parte do escopo do presente trabalho se aprofundar no assunto.

Treinamento por Regra Delta e por Época

A Regra Delta consiste em atualizar os valores das sinapses a cada par entrada-saída que é apresentado. É o método mais direto. A cada par apresentado à rede, executa-se todo o algoritmo de 'Backprogation' e corrige-se os valores sinápticos.

No treinamento por Época, todo o conjunto de treinamento utilizado é apresentado à rede e só após isso, as sinapses são atualizadas. Isto é realizado somando-se os gradientes calculados para cada par entrada-saída até o último par do conjunto de treinamento. Quando passamos todos os pares dizemos que ocorreu uma época. Este tipo de treinamento é mais rápido que por Regra Delta, pois, como podemos facilmente observar a quantidade de operações realizadas é bem menor.

Desenvolvimento

O desenvolvimento deste trabalho pode ser dividido em 5 fases:

Fase 1 → Preparação dos dados;

Fase 2 → Definição da melhor arquitetura da rede para o problema;

Fase 3 → Análise comparativa da performance da rede para diferentes configurações dos parâmetros α(passo de treinamento) e β(momento);

Fase 4 → Verificação de 'Overtraining';

Fase 5 → Análise comparativa da performance no treinamento da rede para diferentes conjuntos de treinamento-teste.

Estas cinco etapas têm por objetivo alcançar a melhor configuração de rede neural para o problema proposto, ou seja, a rede neural que apresente a maior eficiência possível no reconhecimento dos caracteres digitalizados. O software de programação escolhido para a realização deste trabalho foi o MATLAB versão 5.3.1. Foram utilizadas várias funções padrões deste programa, incluindo funções da 'toolbox'(biblioteca de funções) de Redes Neurais versão 3.0.1. A seguir será dada uma breve explicação das funções utilizadas durante o trabalho.

Funções de Redes Neurais utilizadas no MATLAB

`1. newff`

Cria uma rede neural tipo 'feedforward', inicializando as sinapses.

Sintaxe:

```
net=newff(PR,[S1 S2...SN1],{TF1 TF2...TFN1},BTF,BLF,PF)
```

Descrição:

Esta função tem como parâmetros de entrada:

PR → matriz Rx2 dos valores mínimos e máximos dos R elementos de entrada;

Si → tamanho da i-ésima camada, para N1 camadas;

TFi → função de transferência da i-ésima camada, padrão='tansig';

BTF → função de treinamento da rede, padrão='traingdx';

BLF → função de aprendizado dos pesos, padrão='learngdm';

PF → função performance, padrão='mse';

E retorna uma rede de N1 camada tipo 'feedforward'(sem realimentação).

O parâmetro TF foi configurado para ‘tansig’ em todas as camadas. O BTF foi configurado para ‘traingdm’ que será detalhado posteriormente. O BLF e o PF foram deixados com suas configurações padrão.

Uma obervação importante é que a função newff inilicaliza as sinapses da rede. Ela utiliza como padrão para camadas de neurônios tansig a função `initnw` para realizar a inicialização. Esta função é baseada na técnica de Nguyen e Widrow [2] e gera valores iniciais para as sinapses de uma determinada camada de tal modo que as regiões ativas dos neurônios da camada sejam distribuídas uniformemente pelo espaço de entrada. Este método tem várias vantagens sobre a inicialização puramente aleatória dos pesos: (1) poucos neurônios são desperdiçados (sub-utilizados), já que as regiões ativas de todos os neurônios está no espaço de entrada, (2) o treinamento ocorre mais rápido (pois cada área do espaço de entrada possui regiões ativas de neurônios).

2. `train`

Treina uma rede neural.

Sintaxe:

```
[net,tr]=train(NET,P,T,Pi,Ai)
```

Descrição:

Esta função tem como parâmetros de entrada:

NET → o objeto rede neural criado por newff ou outra função de criação;

P → entradas da rede (vetores coluna);

T → saídas alvo (‘targets’) da rede (vetores coluna), padrão=zeros;

Pi → condições iniciais de atraso da entrada, padrão=zeros;

Ai → condições iniciais de atraso por camada, padrão=zeros.

E retorna:

net → nova rede neural;

tr → gravação do treinamento (épocas e performance).

Observar que os parâmetros Pi e Ai são opcionais e não foram utilizados neste trabalho, pois são configurados somente para redes que possuem atrasos nas entradas ou entre camadas.

3. `sim`

Simula uma rede neural.

Sintaxe:

```
[Y,Pf,Af]=sim(net,P,Pi,Ai)
```

Descrição:

A função sim tem como parâmetros de entrada:

net → o objeto rede neural criado por newff ou outra função de criação;

P → entradas da rede (vetores coluna);

Pi → condições iniciais de atraso da entrada, padrão=zeros;

Ai → condições iniciais de atraso por camada, padrão=zeros.

E retorna:

Y → saidas da rede;

Pf → condições finais de atraso da entrada;

Af → condições finais de atraso por camada.

Observar que os parâmetros Pi, Ai, Pf e Af são opcionais e não foram utilizados neste trabalho, pois são configurados somente para que possuem atrasos nas entradas ou entre camadas.

Fase 1 – Preparação dos dados

No início deste trabalho já se dispunha de um banco de dados contendo fotos digitais de placas de veículos. Apesar disto, a quantidade de amostras de letras era insuficiente para um bom treinamento e teste da rede. Para alguns caracteres mais difíceis como o Y, por exemplo, não havia mais de 3 amostras. Fez-se necessário então, a realização de mais fotografias digitais de placas. Devido ao curto espaço de tempo disponível para a coleta de dados, foi escolhida uma quantidade objetivo de 15 amostras de cada caracter para o desenvolvimento do trabalho. Estas 15 amostras serviriam desta forma tanto para formar o conjunto de teste como o de treinamento.

Após a construção deste banco de dados com 15 amostras de cada caracter, o próximo passo foi o tratamento dos dados para serem aplicados à entrada da rede. O figura abaixo mostra o diagrama em blocos do processo de preparação dos dados e um exemplo visual de cada passo.

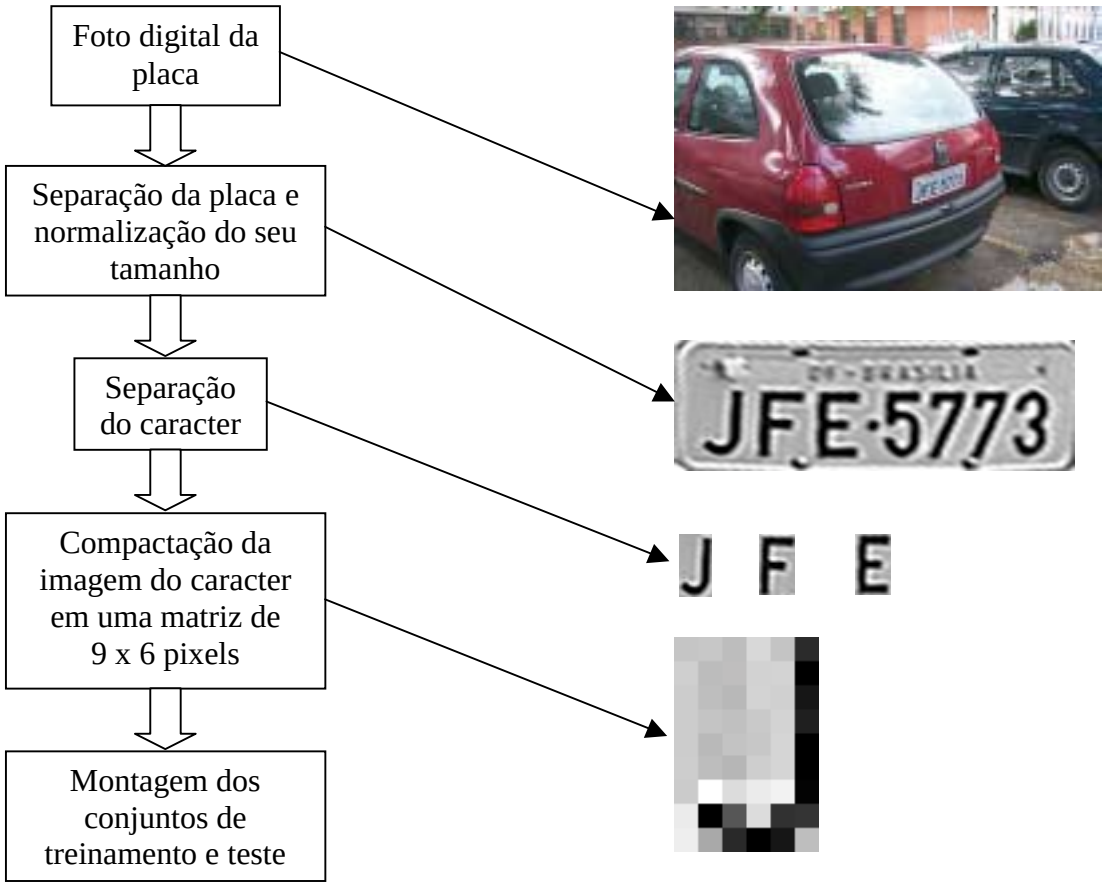


Figura 3: Processo de preparação dos dados

As fotos foram tiradas a uma distância média de 1,5m da traseira do veículo de uma altura em torno de 1,6m do chão (uma pessoa de estatura média). A foto foi tirada em posição frontal à traseira do veículo e não de forma transversal como mostra a figura acima (somente ilustrativa).

O primeiro passo a partir da foto digital foi selecionar somente a placa do veículo da imagem original e converter os pixels para 256 níveis de cinza. Para este padrão, o valor 0 de pixel corresponde ao branco e o valor 255 ao preto. Entre estes dois níveis de cor existem 254 níveis de cinza. Além disso, foi necessário normalizar a placa para ser uma imagem perfeitamente retangular e com um tamanho padrão.

O próximo passo foi separar cada caracter da placa. Este processo foi realizado manualmente com o auxílio de um programa desenvolvido em Java. Foi gerado um arquivo (.txt) de coordenadas para cada caracter da placa.

Como cada caracter possuía, em média, um tamanho de 30 x 20 pixels, o que daria um vetor de 600=30x20 entradas para a rede neural, foi necessário desenvolver um programa para compactar a imagem do caracter em uma matriz de 9 x 6 pixels. Esta

resolução foi escolhida por resultar em um vetor de entrada razoável (54 valores) e por manter a proporção da imagem original (3 x 2). Neste processo de seleção das amostras foi utilizado o programa `montagem.m`, que permitia a visualização amostra por amostra para evitar que se gerasse amostras erradas provenientes de falhas nas etapas anteriores. Ao final desta etapa já se possuía um arquivo ASCII para cada amostra do espaço amostral (26 caracteres). Este arquivo era um arquivo em forma de coluna como realmente deveria ser a entrada da rede no MATLAB como será explicado mais detalhadamente.

Geradas as amostras que iriam ser utilizadas no trabalho, foi necessário montar os conjuntos de treinamento e teste para o treinamento da rede. Para esta tarefa foram escritos três programas. O primeiro programa (`mont87.m`) gera o conjunto de treinamento com 8 amostras e o de teste com 7 amostras. O segundo programa (`mont114.m`) gera o conjunto de treinamento com 11 amostras e o de teste com 4 amostras. O terceiro programa (`mont5conj.m`) gera 5 conjuntos de 3 amostras cada e depois forma o conjunto de treinamento a partir de 3 destes conjuntos e o de teste a partir dos outros 2. Este último programa possui uma diferença básica em relação aos outros dois. Os dois primeiros geravam os conjuntos de maneira ordenada, ou seja, no caso do mont87.m o conjunto de treinamento era formado do seguinte modo: as 8 primeiras colunas consistiam de amostras do caracter A, as próximas 8 colunas eram formadas por caracteres B e assim por diante até o último caracter (Z). O conjunto de teste era formado da mesma forma, só que com 7 amostras. O programa mont5conj.m, devido à forma como foi escrito, ordenava 3 amostras de cada caracter até o último e repetia este processo 3 vezes para o conjunto de treinamento e 2 vezes para o conjunto de teste. Este último programa foi escrito desta forma para que a cada treinamento se pudesse facilmente modificar os conjuntos de treinamento e teste. Com esta última etapa, a fase de preparação dos dados foi concluída. Todos os programas estão anexados a este relatório, inclusive o programa principal `neural.m`.

Fase 2 - Definição da melhor arquitetura de rede

A definição da arquiterura da rede consistiu em determinar 4 características básicas:

1. Número de entradas;
2. Número de camadas de neurônios;
3. Número de neurônios por camada;
4. Função de ativação dos neurônios.

O número de entradas escolhido foi 54, como já foi dito, por ser um nº razoável de entradas e por manter a proporção 3 x 2 do tamanho original. Este número é devido ao número de pixels da matriz correspondente a um caracter compactado (9 x 6). O vetor de entradas de 54 posições é formado colocando-se cada linha da matriz em forma de coluna e empilhando-se uma embaixo da outra. Sendo assim, a entrada da rede consistiu em um vetor coluna de 54 posições de valores variando linearmente de –1 a 1, pois os 256 níveis de cinza antes de serem enviados à rede foram normalizados.

O número de camadas da rede foi definido como 2, já que este número de camadas resolve os problemas de reconhecimento em mais de 95% dos casos.

A configuração de saída escolhida foi a Maximamente Esparsa, onde cada neurônio corresponde a um padrão a ser reconhecido. Isto significa que quando este neurônio está ativo e os demais não, o padrão reconhecido foi o correspondente ao neurônio ativo. Logo, o número de neurônios na camada de saída foi definido como 26.

A função de ativação escolhida foi a tangente hiperbólica, pois para esta função a saída do neurônio pode variar de –1 a 1, permitindo maior eficiência da rede durante o treinamento do que no caso de saídas variando de 0 a 1. Esta função foi usada em todos os neurônios da rede, tanto os da camada intermediária como os da camada de saída.

A obtenção do nº de neurônios ideal na camada intermediária foi realizada em duas etapas. Na primeira etapa foram realizados 15 treinamentos com os mesmos conjuntos de treinamento – teste, onde o nº de neurônios foi sendo alterado de 10 até 50. Os resultados deste teste estão na figura 4 e 5. Como podemos observar na tabela da figura 4, conforme aumentamos a quantidade de neurônios a rede consegue aprender mais padrões. Curioso observar também que a rede aprende 100% alguns padrões e simplesmente ignora outros completamente (0% de eficiência). Este problema foi melhorado posteriormente treinando a rede com outros parâmetros α e β e prolongando-se o treinamento por maior número de épocas.

Outra observação é que a rede teve uma melhora significativa de eficiência quando aumentamos o nº de neurônios de 23 para 25 e depois teve uma queda para 27 neurônios. Este foi um efeito, a princípio sem explicação, mas posteriormente contornado na segunda etapa deste processo, onde fizemos 5 treinamentos com diferentes amostras e traçamos a curva média dos 5 treinamentos.

Eficiência (%)

Padrão/ Nº neurônios	**10**	**13**	**15**	**17**	**20**	**23**	**25**	**27**	**30**	**33**	**35**	**37**	**40**	**45**	**50**
A	100	0	100	0	100	100	100	100	100	100	100	100	100	100	100
B	100	100	100	71	100	100	100	100	100	0	100	100	100	100	100
C	100	71	100	100	100	100	100	100	100	100	100	100	100	100	100
D	71	85	86	71	100	71	86	71	86	86	71	100	86	71	71
E	0	0	100	0	100	100	100	100	100	100	100	100	0	0	100
F	0	0	100	100	100	0	0	0	100	0	100	100	100	100	100
G	57	57	71	57	0	71	71	71	57	71	71	0	71	71	71
H	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
I	0	100	100	100	100	100	100	100	100	100	100	100	100	100	100
J	86	0	0	100	100	100	100	100	100	100	100	100	100	100	100
K	100	100	100	0	86	100	100	0	100	100	100	100	100	100	100
L	0	100	100	100	100	0	100	100	0	100	100	100	100	100	100
M	100	71	86	86	100	86	86	86	0	100	86	100	86	86	86
N	86	100	100	100	100	100	100	100	100	100	100	100	100	100	100
O	57	86	71	57	57	86	86	71	0	43	71	57	57	57	57
P	85	100	100	0	100	100	100	0	100	100	100	100	100	100	100
Q	100	100	0	100	100	100	100	100	100	100	100	100	100	100	100
R	0	100	100	100	100	100	100	100	100	100	100	100	100	100	0
S	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100
T	0	100	86	100	86	0	100	100	100	100	100	100	100	100	100
U	0	100	86	100	0	100	100	100	100	100	100	100	100	100	100
V	100	100	100	100	100	100	100	86	100	100	0	100	100	100	100
X	86	86	100	100	0	100	86	100	86	100	86	86	100	86	100
Y	0	86	57	86	86	86	86	86	86	86	86	86	86	86	86
W	0	86	71	86	86	86	86	86	86	86	86	100	86	100	100
Z	100	100	0	100	100	100	100	100	100	100	100	86	100	100	100

Figura 4: Percentual de eficiência (%) para treinamentos com diferente número de neurônios

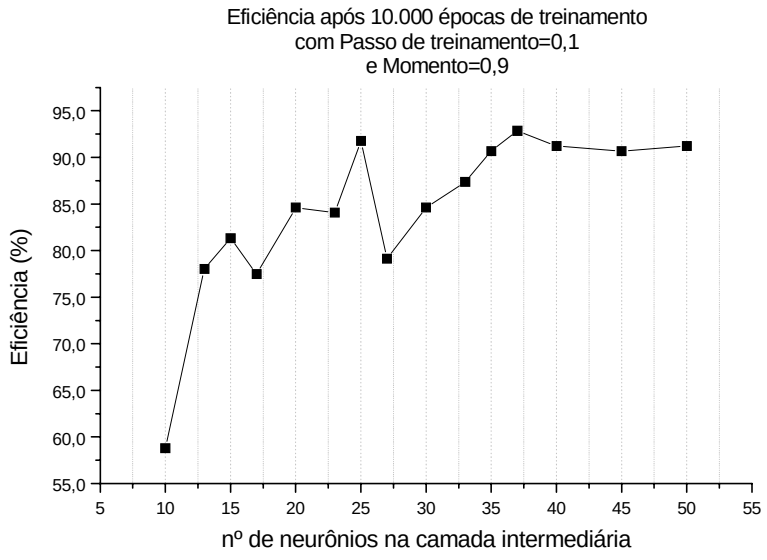


Figura 5: Eficiência média com diferentes nº de neurônios na camada intermediária

O gráfico abaixo mostra o tempo de processamento dos treinamentos realizados acima. Todos eles foram realizados em um computador tipo PC equipado com um Pentium II 333MHz e 32MB de memória.

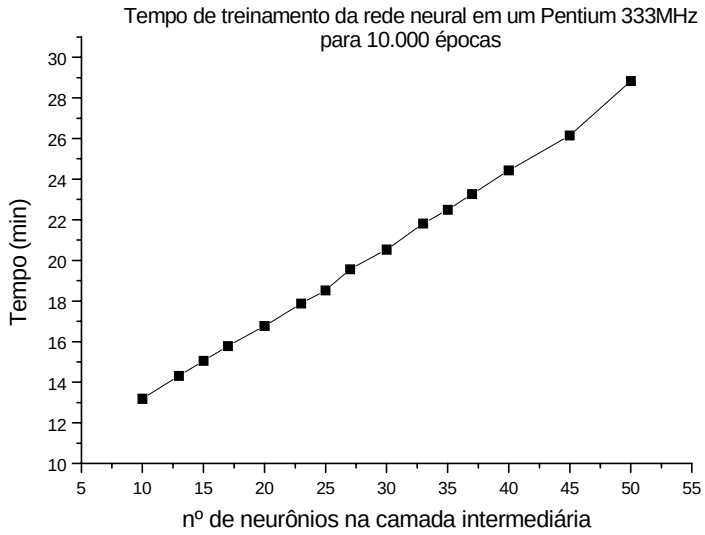


Figura 6: tempo de processamento de vários treinamentos

Na segunda etapa do processo para se determinar o número “ótimo” de neurônios na camada intermediária, realizamos 5 treinamentos com diferentes configurações de conjuntos treinamento – teste. Isto foi feito para que a decisão tomada não fosse dependente dos conjuntos utilizados. Ao final de cada treinamento, foi traçada a curva de eficiência da rede. Após os 5 treinamentos, foi traçada a média das 5 curvas.
A partir dos resultados, foram escolhidos 35 neurônios como a configuração que apresentou a melhor eficiência nos cinco testes e obviamente também na média. Para todos os treinamentos, o critério de parada foi deixar o algoritmo convergir, ou seja, treinar a rede até que a curva de performance (MSE) apresentasse uma taxa de variação muito baixa.
Resumo da melhor arquitetura da rede encontrada:

- 54 entradas;
- 2 camadas de neurônios;
- 35 neurônios na camada intermediária;
- 26 neurônios na camada de saída;
- função de ativação = tan. hiperbólica para todos os neurônios.

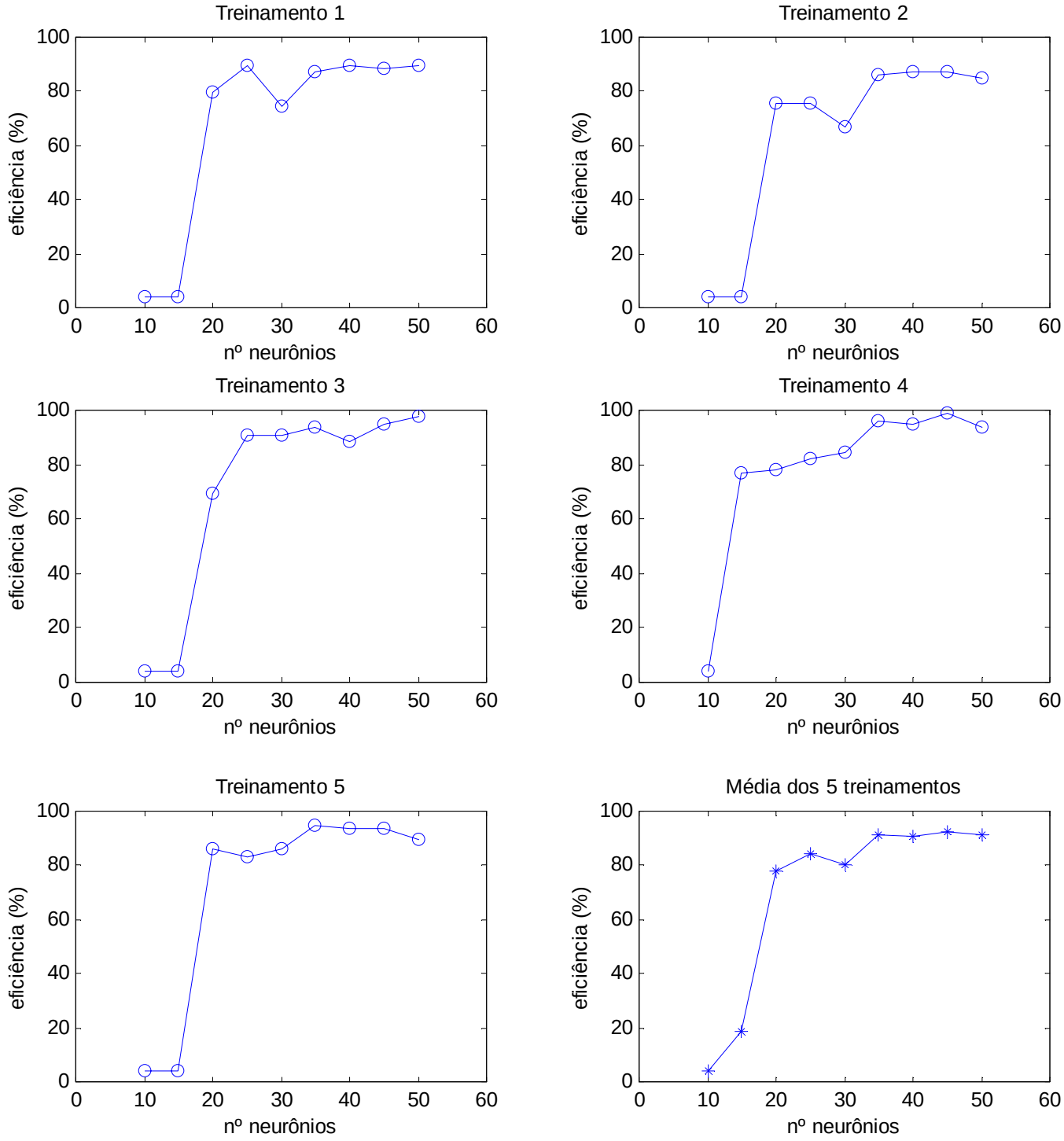


Figura 7: Eficiência para diferente nº de neurônios

Fase 3 - Análise comparativa da performance da rede para diferentes configurações dos parâmetros α(passo de treinamento) e β(momento)

Para a determinação dos valores "ótimos" de α e β foram realizados 16 treinamentos, utilizando a configuração de rede definida na seção anterior para os seguintes valores: $\alpha \in \{0.01, 0.1, 0.5, 0.5\}$ e $\beta \in \{0, 0.1, 0.5, 0.9\}$. Os testes foram realizados usando 8 amostras para treinamento. Os resultados são mostrados abaixo.

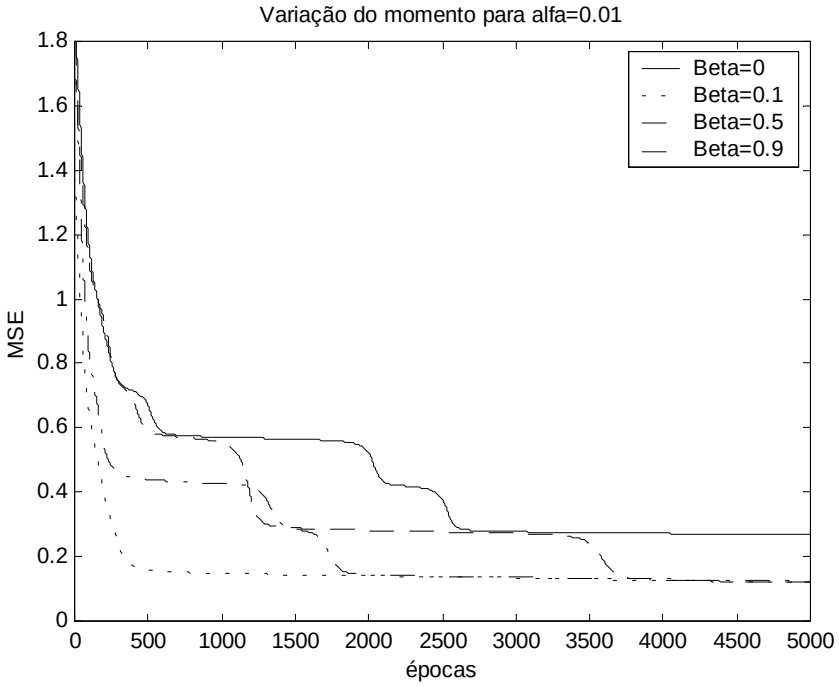


Figura 8: Determinação dos parâmetros "ótimos" - Alfa=0.01

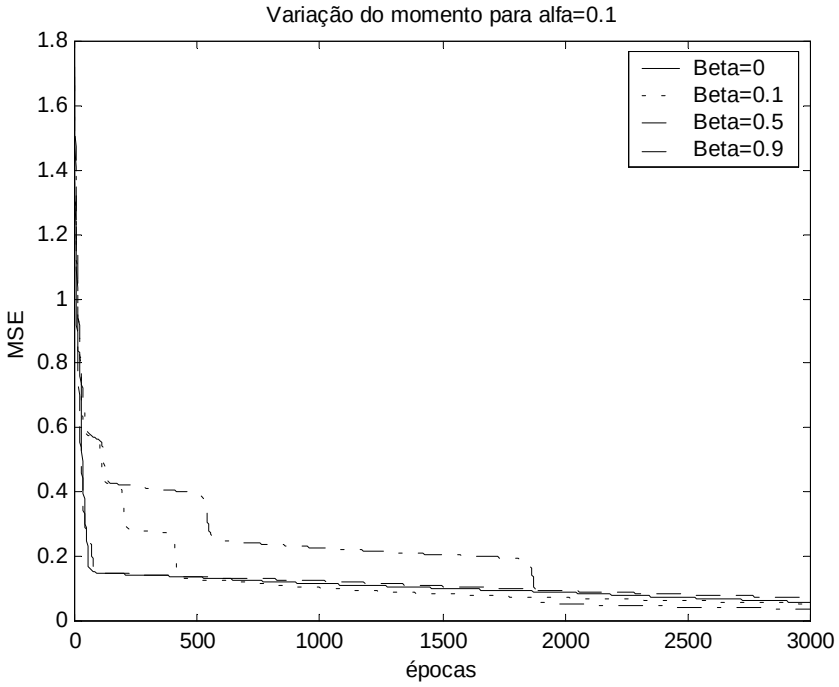


Figura 9: Determinação dos parâmetros "ótimos" - Alfa=0.1

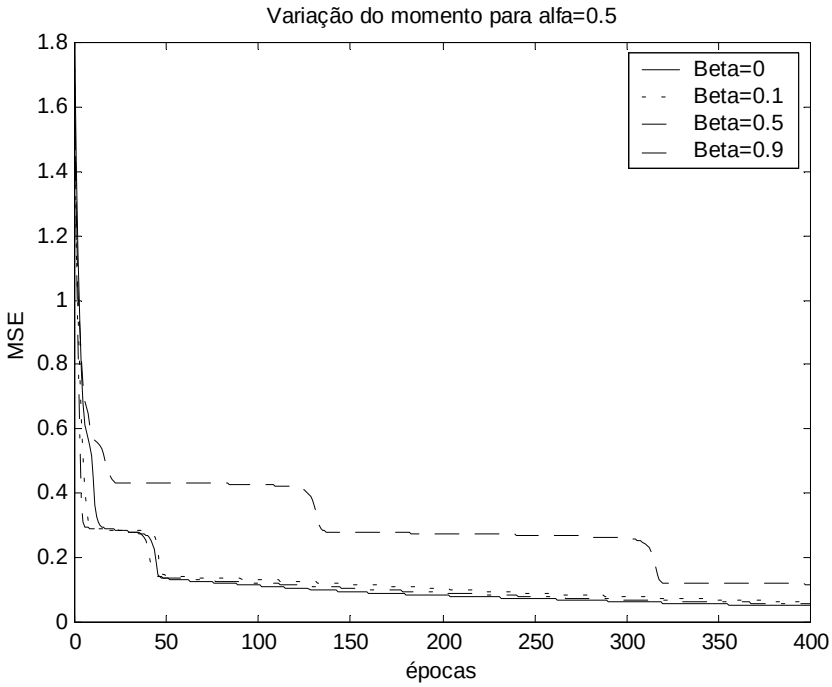


Figura 10: Determinação dos parâmetros “ótimos” - Alfa=0.5

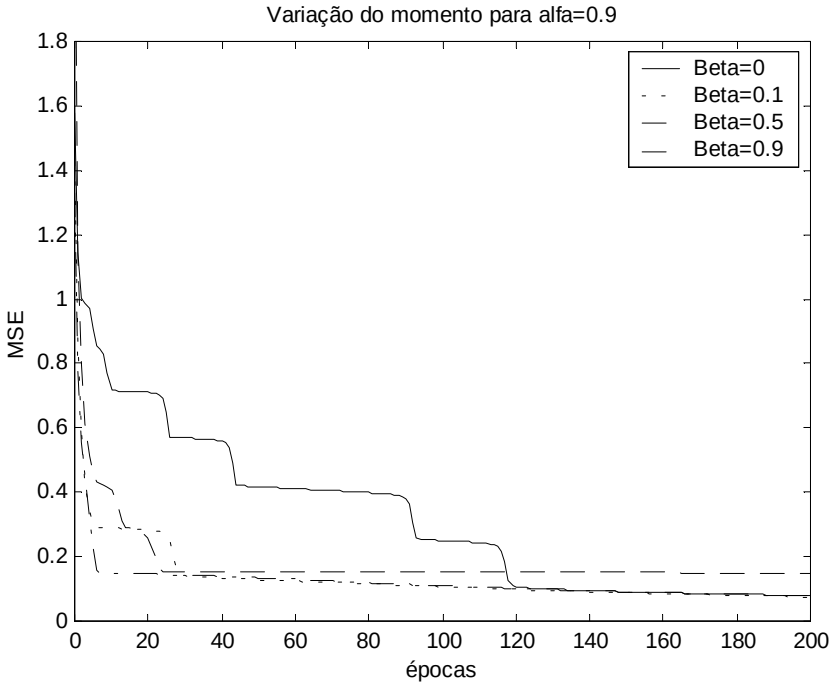


Figura 11: Determinação dos parâmetros “ótimos” - Alfa=0.9

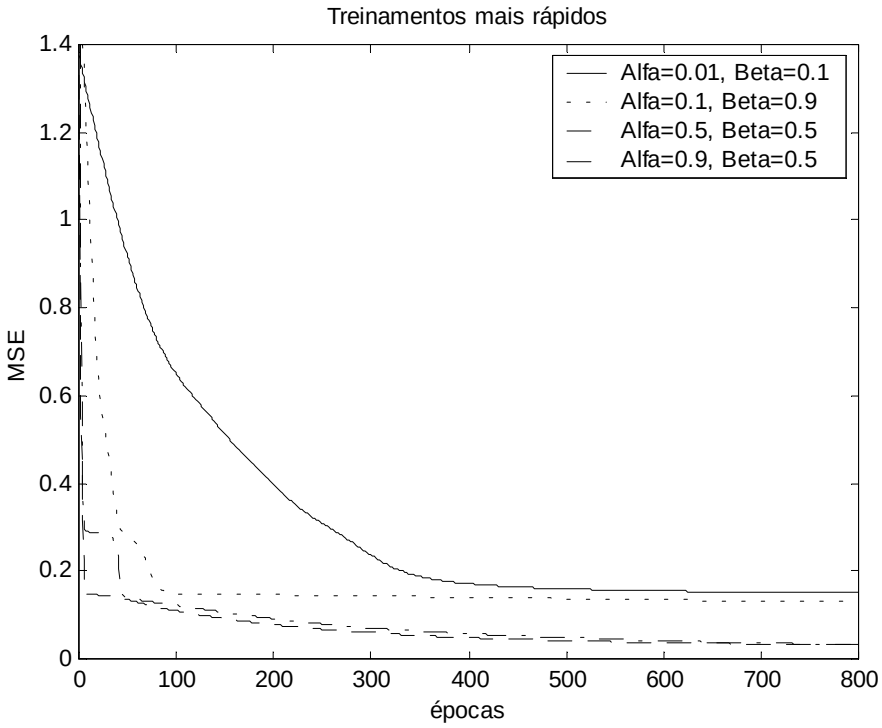


Figura 12: Determinação dos parâmetros “ótimos” – Melhores curvas

As curvas de treinamento variando-se os parâmetros α e β sugerem as seguintes conclusões:

- Quanto menor o passo de treinamento α, mais lenta é a convergência do algoritmo, o que é previsto pela teoria, e quanto maior o α, mais rápida é a convergência. Enquanto no primeiro gráfico, com α=0.01, a convergência de todas as curvas só ocorre para 4000 épocas, no último gráfico, com α=0.9, para 140 épocas todos os treinamentos já haviam convergido;
- Para um mesmo valor de α, o algoritmo levou diferentes nº de épocas para convergir, o que ocorreu em todos os cinco gráficos. Isto é explicado pelo ponto de partida do treinamento. Para cada treinamento, as sinapses eram inicializadas com valores aleatórios diferentes, o que define um ponto de partida na superfície de erro do problema diferente. Ou seja, em alguns treinamentos, o ponto de partida estava mais próximo de um mínimo local enquanto em outros estava mais distante;
- Apesar de levar o valor de α até 0.9, o algoritmo não demonstrou oscilações, como poderia ocorrer. Isto pode se dever ao fato da superfície de erro do problema ter um comportamento muito ‘suave’, isto é, possuir inclinações suaves;

- A variação do parâmetro β (momento) não mostrou nenhuma influência no comportamento das curvas, já que não se observou nenhum caso de oscilação. Caso isto ocorresse, provavelmente, seria para um alto valor de α com β tendendo a 0, já que β é um fator que melhora a estabilidade de performance do algoritmo;
- Observando-se o último gráfico, que mostra as curvas que convergiram mais rápido dos quatro gráficos anteriores, chegamos à conclusão que os melhores parâmetros para este problema são α=0.9 e β=0.5, que levou à convergência mais rápida e ao menor MSE final.

Fase 4 - Verificação de 'Overtraining'

Foram realizados 3 treinamentos para a verificação se ocorreria a especialização excessiva da rede com perda de generalização, fenômeno conhecido como 'Overtraining'. Os testes foram feitos da seguinte forma: a cada 100 épocas de treinamento, passava-se o conjunto de teste pela rede treinada e calculava-se o MSE da saída. Ao final, foram traçados gráficos do MSE do conjunto de treinamento versus o MSE do conjunto de teste. Caso ocorresse o 'overtraning', a partir de certo ponto, a performance do conj. de teste começaria a piorar enquanto do conj. de treinamento continuaria a melhorar. Não se observou o fenômeno para os três treinamentos, o primeiro até 5000 épocas, o segundo até 20000 épocas e o último até 40000 épocas. Os gráficos são mostrados abaixo para 8 amostras de treinamento, α=0.1 e β=0.9.

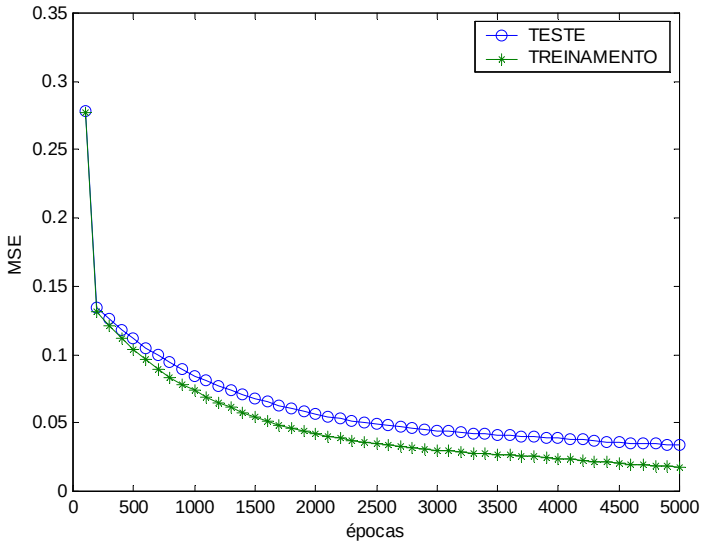


Figura 13: Verificação da inexistência de 'Overtraining' até 5000 épocas

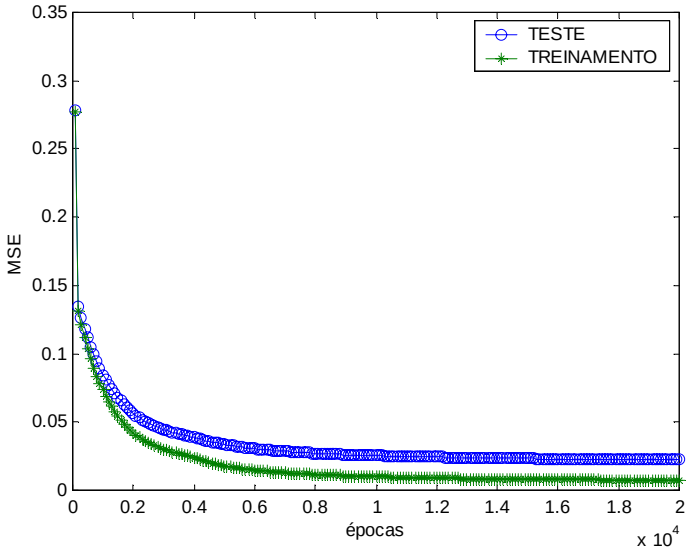


Figura 14: Verificação da inexistência de 'Overtraining' até 20000 épocas

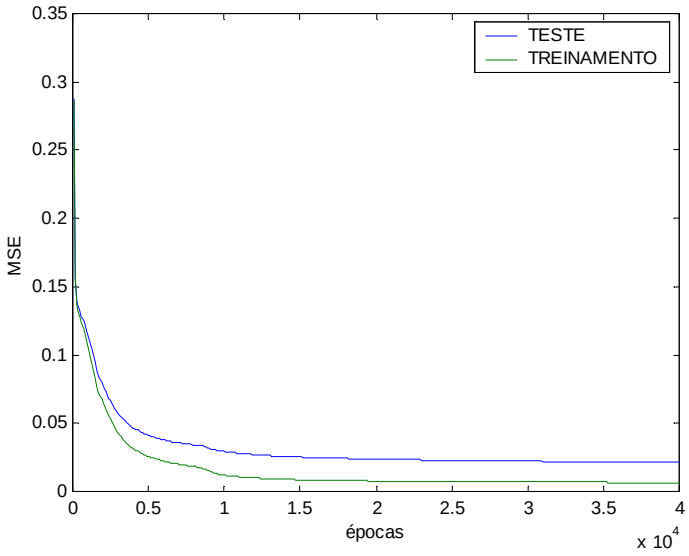


Figura 15: Verificação da inexistência de 'Overtraining' até 40000 épocas

Fase 5 - Análise comparativa da performance no treinamento da rede para diferentes conjuntos de treinamento-teste

Foram feitos dezenas de treinamentos com diferentes conjuntos de treinamento – teste para se verificar a sensibilidade da rede à mudança dos pares entrada-saída no aprendizado. Destes testes foram selecionados alguns resultados, com os respectivos MSE e Eficiência resultantes do treinamento, mostrados abaixo.

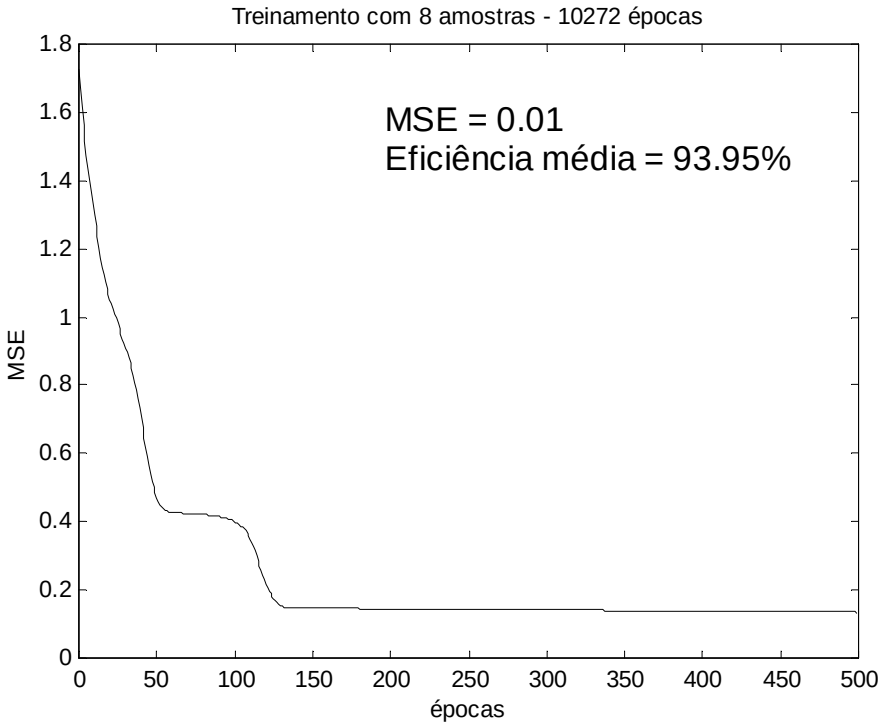


Figura 16: Teste da sensibilidade da rede ao conjunto de treinamento – 8 amostras

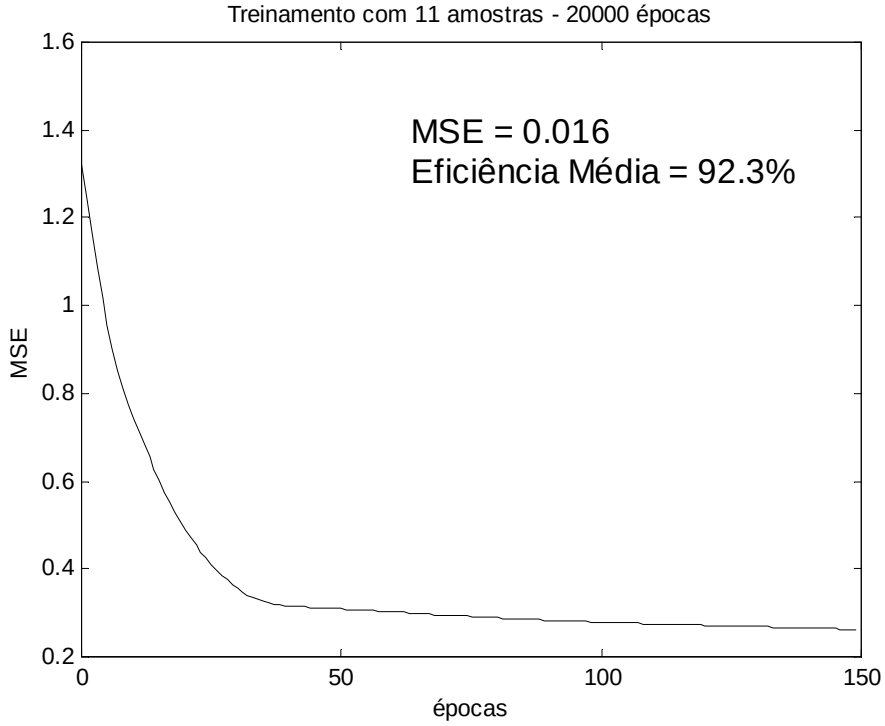


Figura 17: Teste da sensibilidade da rede ao conjunto de treinamento – 11 amostras

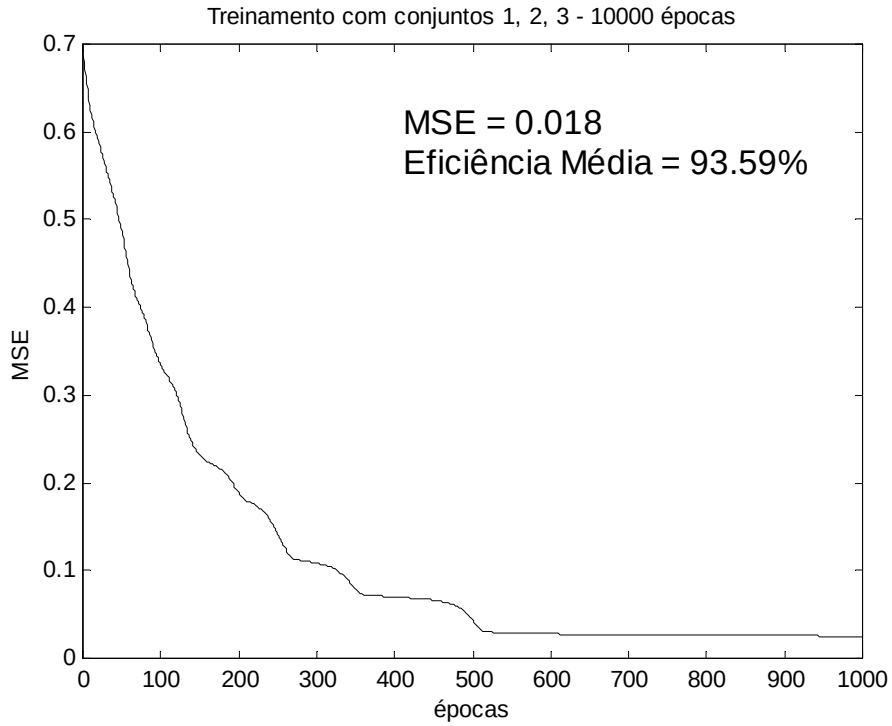


Figura 18: Teste da sensibilidade da rede ao conjunto de treinamento – 9 amostras

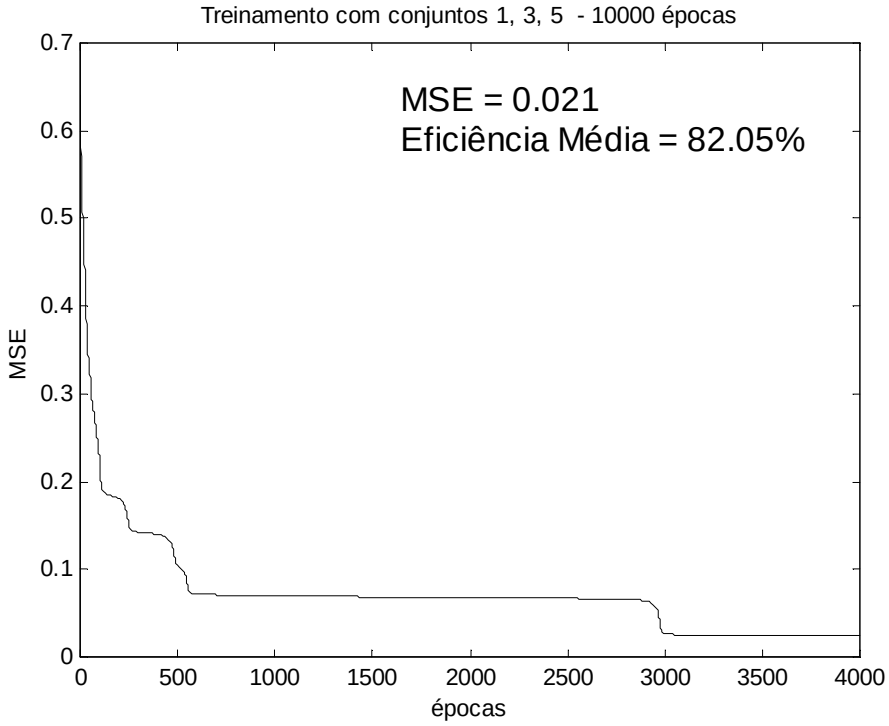


Figura 19: Teste da sensibilidade da rede ao conjunto de treinamento – 9 amostras

Dos resultados dos mostrados acima, podemos concluir que a rede não se mostrou muito sensível nem ao nº de amostras que formavam os conjuntos treinamento – teste e nem quais amostras eram utilizadas. O fato do treinamento com 11 amostras ter convergido mais rápido que os demais não quer dizer que com mais amostras o treinamento converge mais rápido, pois foram feitos outros treinamentos que demonstraram que este efeito nem sempre ocorre. A explicação reside no ponto de partida na superfície do erro. Já a constância das eficiências demonstrou claramente que a mudança nos conjuntos treinamento – teste não influenciaram em praticamente nada na performance final da rede.

Conclusões

Ao longo do desenvolvimento deste trabalho foram realizados inúmeros treinamentos visando chegar à melhor configuração de rede neural para o problema de reconhecimento dos caracteres. Dos resultados expostos podemos tirar as seguintes conclusões:

- Para o reconhecimento dos 26 padrões apresentados, dispondo de 15 amostras de cada um, a rede neural de duas camadas necessitou de, no mínimo, 35 neurônios na camada intermediária para alcançar uma eficiência muito boa (≈ 95%);
- A utilização dos parâmetros ‘ótimos’ α=0.9 e β=0.5 levou o treinamento da rede à convergência mais rápida devido ao alto valor do passo de treinamento e não causou oscilação da performance durante o treinamento;
- Para os conjuntos de treinamento e teste disponíveis a rede não demonstrou o fenômeno de ‘Overtraining’ até 40000 épocas;
- A rede não se mostrou sensível à variação das amostras e nem à quantidade utilizada nos conjuntos de treinamento e teste como foi mostrado pelos resultados da fase 5. A convergência mais rápida em alguns treinamentos pode ser explicada pela inicialização das sinapses.

A melhor rede neural obtida utilizou 54 entradas, 35 neurônios na camada intermediária, 26 na camada de saída, α = 0.9, β=0.5, 10.000 épocas de treinamento, 9 amostras para treinamento, 6 para teste e apresentou os seguintes resultados:

Padrão	Eficiência (%)	Padrão	Eficiência (%)	Padrão	Eficiência (%)	Padrão	Eficiência (%)	Padrão	Eficiência (%)
A	100	G	66.6	M	66.6	S	100	W	100
B	100	H	100	N	100	T	100	Z	100
C	100	I	100	O	100	U	100		
D	100	J	100	P	100	V	100		
E	100	K	100	Q	100	X	100		
F	100	L	100	R	100	Y	100		

MSE = 0.0035

Eficiência média = 97.4%

REFERÊNCIAS BIBLIOGRÁFICAS

1. *Neural Networks – A comprehensive foundation*, Simon Haykin, Macmillan College Publishing Company, New York, 1994.

2. *Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights*, Nguyen, D., and B. Widrow, Proceedings of the International Joint Conference on Neural Networks, vol 3, pp. 21-26, 1990.

3. *Introdução às Redes Neurais*, Caloba, Notas de Aula, 2000.

4. *Neural Network Toolbox – For use with MATLAB*, Howard Demuth and Mark Beale, User's Guide-Version 3.0, Cap. 5, The MathWorks, Inc., 1998.

ANEXO

- Listagem dos programas -

```
% ************************************************************************
% ***    Programa de Redes Neurais - Neural.m                          ***
% ***    - cria, treina e simula uma Rede 'FeedForward' de duas camadas -  ***
% ***    Herman Lima Jr                                                ***
% ***    última atualização: 28/06/2000                                ***
% ************************************************************************
%
limp = input('Limpar todas as variáveis? (s,n) ','s');
if limp=='s'
   clear all
   close all
end
%
% Definições da Arquitetura da Rede
n_ent = 54;             % nº de entradas
n_inter = 25;           % nº de neuronios na camada intermediária
n_saida = 26;           % nº de neuronios na camada de saída
tipo_inter='tansig';    % tipo de neuronios da camada intermediária
tipo_saida='tansig';    % tipo de neuronios da camada de saída
tipo_trein='learngd';   % tipo de treinamento da rede
%
% ****************************************************
% *** Montagem dos conjuntos de treinamento e teste ***
% ****************************************************
exec = input('Montar conjuntos de treinamento e teste? (s,n) ','s');
if exec=='s'
   [p,t,test] = mont5conj;             % Treinamento (8 amostras), Teste (7
amostras)
   disp('CONJUNTOS DE TREINAMENTO E TESTE MONTADOS');
   [pn,minp,maxp] = premnmx(p);     % Normalizando o conj. de treinamento para [-
1:1]
   tn = t;
   [testn,mintest,maxtest] = premnmx(test);  % Normalizando o conj. de teste para
[-1:1]
end
%
% ****************************
% *** Criando a Rede Neural ***
% ****************************
exec = input('Criar uma nova rede neural? (s,n) ','s');
if exec=='s'
   net = newff([minp maxp],[n_inter,n_saida],{tipo_inter,tipo_saida},'traingdm');
end
%
% ******************************
% *** Treinando a Rede Neural ***
% ******************************
exec = input('Treinar a rede neural? (s,n) ','s');
if exec=='s'
   erro = input('Entre com o erro (MSE) desejado: ');
   % Configurando parâmetros para treinamento por REGRA DELTA
   net.biases{1,1}.learnFcn = tipo_trein;
   net.biases{2,1}.learnFcn = tipo_trein;
   net.layerWeights{2,1}.learnFcn = tipo_trein;
   net.inputWeights{1,1}.learnFcn = tipo_trein;
   net.layerWeights{2,1}.learnParam.lr = 0.1;   % Passo de treinamento (ALFA)
   net.layerWeights{2,1}.learnParam.lr = 0.1;   % Passo de treinamento (ALFA)
   net.adaptParam.passes = 200;  % Ciclos de treinamento
   %
   % Configurando parâmetros para treinamento por ÉPOCA COM MOMENTO
   net.trainParam.show = 100;
   net.trainParam.lr = 0.2;      % Passo de treinamento (ALFA)
   net.trainParam.mc = 0.9;      % Momento (BETA)
   net.trainParam.goal = erro;
```

```
   net.trainParam.epochs = 1000;
   %
   tipo = input('Treinamento por Regra Delta (d) ou Época com momento (e)?
','s');
   if tipo=='d'
      tic
      [net,y,e] = adapt(net,pn,tn);       % Treinamento REGRA DELTA
      Tempo_de_treinamento = toc
   else
      %for i=1:50
         %i
         [net,tr] = train(net,pn,tn);     % Treinamento ÉPOCA COM MOMENTO
         %trn_erro(i) = tr.perf(100);
         %saida = sim(net,testn);
         %erroatual = saida - outtest;
         %tst_erro(i) = mse(erroatual);
      %end
   end
   %x = (100:100:5050);
   %plot(x,tst_erro,'o-',x,trn_erro,'*-');
   %legend('TESTE','TREINAMENTO');
   %xlabel('épocas');
   %ylabel('MSE');
end
%
% ******************************
% *** Simulando a Rede Neural ***
% ******************************
exec = input('Simular a rede neural? (s,n) ','s');
if exec=='s'
   saida = sim(net,testn);
   %ef = efic(saida,7);    % Calculando a eficiência (MODIFICAR O Nº DE AMOSTRAS)
   ef = efic_2(saida);     % Calculando a eficiência para mont5conj.m
end
%
% ***********************************
% *** Salvando os resultados (MSE) ***
% ***********************************
exec = input('Gravar resultados do treinamento e simulação? (s,n) ','s');
if exec=='s'
   nt = input('Entre com o número do treinamento: ');
   nt = num2str(nt);
   %
   arq_perf = strcat('trperf',nt);
   fid = fopen(arq_perf,'w');
   fprintf(fid,'%12.8f\n',tr.perf);
   fclose(fid);
   %
   arq_epoch = strcat('trepoch',nt);
   fid = fopen(arq_epoch,'w');
   fprintf(fid,'%12.8f\n',tr.epoch);
   fclose(fid);
   %
   arq_ef = strcat('efic',nt);
   fid = fopen(arq_ef,'w');
   fprintf(fid,'%12.8f\n',ef);
   fclose(fid);
end
```

```
% ************************************************************************
% ***    Programa de Redes Neurais - mont87.m                          ***
% ***    - monta os conjuntos de Treinamento-Teste no esquema 8-7 -    ***
% ***    Herman Lima Jr                                                ***
% ***    última atualização: 06/06/2000                                ***
% ************************************************************************
%
function [intrain,outtrain,intest,outtest] = mont87()
%
% Montagem do Conjunto de Treinamento
let =
['a';'b';'c';'d';'e';'f';'g';'h';'i';'j';'k';'l';'m';'n';'o';'p';'q';'r';'s';'t';'
u';'v';'x';'y';'w';'z'];
outtrain = (-1)*ones(26,26*8);
for letra=1:26
   for ind=1:8          % Amostras 1 a 8
      aux=strcat(let(letra),num2str(ind));
      aux=strcat(aux,'.txt');
      intrain(:,ind+((letra-1)*8))=load(aux);
      outtrain(letra,ind+((letra-1)*8))=1;
   end
end
%
% Montagem do Conjunto de Teste
outtest = (-1)*ones(26,26*7);
for letra=1:26
   for ind=9:15         % Amostras 9 a 15
      aux=strcat(let(letra),num2str(ind));
      aux=strcat(aux,'.txt');
      intest(:,(ind-8)+((letra-1)*7))=load(aux);
      outtest(letra,(ind-8)+((letra-1)*7))=1;
   end
end
```

```
% ************************************************************************
% ***    Programa de Redes Neurais - mont114.m                         ***
% ***    - monta os conjuntos de Treinamento-Teste no esquema 11-4 -   ***
% ***    Herman Lima Jr                                                ***
% ***    última atualização: 10/06/2000                                ***
% ************************************************************************
%
function [intrain,outtrain,test] = mont114()
%
% Montagem do Conjunto de Treinamento
let =
['a';'b';'c';'d';'e';'f';'g';'h';'i';'j';'k';'l';'m';'n';'o';'p';'q';'r';'s';'t';'
u';'v';'x';'y';'w';'z'];
outtrain = (-1)*ones(26,26*8);
for letra=1:26
   for ind=1:11         % Amostras 1 a 11
      aux=strcat(let(letra),num2str(ind));
      aux=strcat(aux,'.txt');
      intrain(:,ind+((letra-1)*11))=load(aux);
      outtrain(letra,ind+((letra-1)*11))=1;
   end
end
%
% Montagem do Conjunto de Teste
for letra=1:26
   for ind=12:15        % Amostras 12 a 15
      aux=strcat(let(letra),num2str(ind));
      aux=strcat(aux,'.txt');
      test(:,(ind-11)+((letra-1)*4))=load(aux);
   end
end
```

```
% ************************************************************************
% ***    Programa de Redes Neurais - mont5conj.m                       ***
% ***    - monta 5 conjuntos de amostras escolhidas aleatoriamente  -  ***
% ***    Herman Lima Jr                                                ***
% ***    última atualização: 10/06/2000                                ***
% ************************************************************************
%
function [intrain,outtrain,test] = mont5conj()
%
conjtrain = [1 2 3];       % conjuntos de treinamento - MODIFICAR P/CADA
TREINAMENTO !!
conjtest = [4 5];          % conjuntos de teste - MODIFICAR P/CADA TREINAMENTO !!
let =
['a';'b';'c';'d';'e';'f';'g';'h';'i';'j';'k';'l';'m';'n';'o';'p';'q';'r';'s';'t';'
u';'v';'x';'y';'w';'z'];
outtrain = (-1)*ones(26,26*8,5);
%
% Montagem dos Conjuntos de Treinamento
for conj=1:5
   conj
   for ind=1:3
      for letra=1:26
         aux = strcat(let(letra),num2str(ind+((conj-1)*3)));
         aux = strcat(aux,'.txt');
         intrainaux(:,ind+((letra-1)*3),conj) = load(aux);
         outtrainaux(letra,ind+((letra-1)*3),conj) = 1;
      end
   end
end
intrain = [intrainaux(:,:,conjtrain(1)) intrainaux(:,:,conjtrain(2))
intrainaux(:,:,conjtrain(3))];
outtrain = [outtrainaux(:,:,conjtrain(1)) outtrainaux(:,:,conjtrain(2))
outtrainaux(:,:,conjtrain(3))];
test = [intrainaux(:,:,conjtest(1)) intrainaux(:,:,conjtest(2))];
%save intrain intrain -ascii
%save outtrain outtrain -ascii
```

```
% ********************************************************
% ***    Programa de Redes Neurais - montagem.m          ***
% ***    - visualiza e gera os padrões usando IMRESIZE -  ***
% ***    Herman Lima Jr                                  ***
% ***    última atualização: 28/06/2000                  ***
% ********************************************************
%
close all
clear all
%
letramin = input('Entre com a letra minúscula:  ','s');
fid = fopen('nomes','r');
nomes = fscanf(fid,'%s',[8,57])';          % le arquivo com os nomes das placas
iptsetpref('ImshowTruesize','manual');     % mostra imagem com autoescala
Xnovo = 6;Ynovo = 9;                       % tamanho da imagem apos o
redimencionamento
loop = 0;
i = 0;
cont = 0;
for j=1:455
   for k=1:3
      teste = nomes(j,k);
      if teste==letramin
         cont = cont + 1;
      end
   end
end
disp('Número de amostras: ');
disp(cont);
while loop~=15,
   i = i + 1
   nome_arq = strcat(nomes(i,1:8));          % le nome arquivo imagem
   nome_txt = strcat(nomes(i,:),'.txt');     % insere extensao .txt a nomes
   [X,map] = tiffread(nome_arq);             % le imagem .tif
   coord = load(nome_txt);                   % le arquivo com as coordenadas dos
caracteres
   for ind=1:4:12,
      teste = nomes(i,round(sqrt(ind)));
      if teste==letramin
         X2 = X(coord(ind+1):coord(ind+3),coord(ind):coord(ind+2));
         X2 = imresize(X2,[12 8]);
         imshow(X2,map);
         salva = input('Salvar arquivo ou sair? (y,n,s): ','s');
         if salva=='s'
            loop = 15;
         end
         if salva=='y'
            loop = loop + 1
            indarq = num2str(loop);
            arq = strcat(letramin,indarq);
            arq = strcat(arq,'.txt');
            a = fopen(arq,'w');
            fprintf(a,'%12.8f\n',X2);
            fclose(a);
         end
      end
   end
end
%
close all
clear all
```